Multilingual Named Entity Recognition

A Thesis Presented

by

Vivek Vasant Kulkarni

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

May 2014

Stony Brook University

The Graduate School

Vivek Vasant Kulkarni

We, the thesis committee for the above candidate for the

Master of Science degree, hereby recommend

acceptance of this thesis

Prof.Steven Skiena – Thesis Advisor
Distinguished Teaching Professor, Department of Computer Science

Prof.I.V.Ramakrishnan
Professor, Department of Computer Science

Prof.Leman Akoglu
Assistant Professor, Department of Computer Science

Prof.Yejin Choi
Assistant Professor, Department of Computer Science

This thesis is accepted by the Graduate School.

Charles Taber
Dean of the Graduate School

Abstract of the Thesis

Multilingual Named Entity Recognition

by

Vivek Vasant Kulkarni

Master of Science

in

Computer Science

Stony Brook University

2014

With the massive amounts of unannotated text available from myriad sources, learning representations useful for natural language processing(NLP) tasks is an increasingly popular research area. Using deep learning techniques, we learn distributed representations for words (word embeddings) using Wikipedia as the source of text for 40 languages. These distributed representations represent each word as a point in feature space and capture useful semantic and syntactic properties of words amd have been shown to be useful in NLP Tasks like Part of Speech Tagging(POS) etc. We have built 2 classes of word embeddings namely Polyglot and Skipgram for these languages.

We build a named entity recognition (NER) system that supports 40 languages using the word embeddings we have generated as features and seek to use freely available Wikipedia text as training data. This involves training language models to obtain the word embeddings, understanding the properties of the learnt word embeddings and culminates in learning models for named entity classification. We also present a novel technique for evaluating our performance on the myriad languages for which no gold data set for testing exists. Our results demonstrate that word embeddings exhibit nice community structure and can be used effectively for NER with no explicit hand crafted feature engineering and perform competitively with existing baselines when coupled with simple language agnostic techniques.

Dedicated to my parents.

Contents

List of Figures

List of Tables

Acknowledgements

My stint at Stony Brook University to attend graduate school have been two very fulfilling years of my life. I have had the opportunity of learning how to do research and met some very talented people during the process. I have also had the opportunity of meeting people from different walks of life which has no doubt helped me grow as a individual.My interactions with faculty and students from the life sciences, sociology and neuroscience have deepened by fascination for science and rekindled the childlike curiosity I once had. I wish to thank everyone of them without which this thesis would never be written.

I would like to thank my advisor Professor Steven Skiena for gently guiding me all through the process and providing me his constant guidance and encouragement to pursue research. I would also like to thank all the members of the Data Science Lab for their support and encouragement especially when I was feeling low(read *dead ends in research*) - Rami Al-Rfou', Bryan Perozzi, Vishwas Sharma, Moutupsi Paul, Abhinav Gupta. I would also like to thank all my friends in graduate school especially Jagat Sastry, Alok Katiyar, Sagardeep Mahapatra, Sagar Trehan and Abhradeep Guha Thakurta for helping me keep my sanity all through graduate school.

Finally I owe all of this to my family, who believed in my abilities and without whose love and support I would never have taken the plunge into graduate school.

Chapter 1

Introduction

1.1 Distributed Word Representations

Traditionally most applications in natural language processing, have revolved around learning models from annotated data (training data) with explicit feature engineering geared towards the task at hand. Presently, given the massive amount of unannotated text floating around on the internet and other sources, the question of whether one can use computers to automatically learn features and representations from this text useful for natural language processing tasks becomes increasingly relevant. One approach that explores this problem and has recieved a lot of popular media attention is the technique of deep learning. Deep Learning seeks to learn useful feature representations by utilizing deep neural networks. Deep Neural Networks learn features corresponding to different levels of abstractions. Thus for natural language processing tasks we seek to learn representations of words (*word embeddings*). Some useful features of these word embeddings are highlighted below(see [1] for more details):

- **Distributed Representations**: Word embeddings are distributed representations in the sense that each feature is not mutually exclusive of the other. The fact that the representation learnt is distributed helps deal with the curse of dimensionality.

- **Continuous Valued Features**: Word embeddings are continuous valued. Thus each word can be thought of as representing a point in the feature space. This is also known to make the optimization problem in learning much simpler.

- **Meaningful clusters of words**: We hope that the neural network learns word representations such that functionally or syntactically similar words are close to each other in feature space. This is typically achieved by different models by choosing a suitable loss function by usually noting that similar words in the context can be replaced by one another.

We have built 2 word embeddings namely, Polyglot and Skipgram for 40 languages. These embeddings are learnt by using Wikipedia text as the data source. A detailed description of the training procedure is described in Chapter 3.

1.2 Multilingual Named Entity Recognition using word embeddings

We seek to build a Named Entity Recognition system (NER) for 40 languages using word embeddings that were learnt from Wikipedia texts for these 40 languages. In order to build this system, we attacked the following subproblems in order:

1. **Improving performance of training language models** Since training language models is a time consuming task, we investigated how we can train these language models faster. We investigated whether it is beneficial to use GP-GPU's for speedening up the training of these language models.

2. **Investigate the properties of Polyglot and Skipgram Embeddings** In order to gain insight into how the different models differ in the properties of the word representations learnt, we investigated the network structure of Polyglot and Skipgram embeddings. This could help us gain insights into the choice of the embeddings to be used for a particular NLP task.

3. **Multilingual Named Entity Recognition** Having set the stage, and gained insight into properties of Polyglot embeddings, we finally use Polyglot Embeddings to build a NER system for 40 languages.

1.3 Thesis Outline

Chapter 2 explores the problem of training language models on GP-GPU's and outlines our findings and conclusions.Chapter 3 explores the properties of both Polyglot and Skipgram embeddings by inducing a network over them and studying their network structure.Chapter 4 describes our system for multilingual NER for 40 languages.In Chapter 5 we conclude by outlining future work and further research directions. Chapters 2, 3 and 4 are joint work with Rami Al Rafou' and Bryan Perozzi. Chapter 3 and Chapter 4 are publications and hence appear as is (except for minor modifications to suit thesis requirements).

Chapter 2

Using GPU's to train Deep Belief Networks

2.1 Introduction

GPU computing has been of significant interest to the research community. This has led to the evolution of heterogeneous parallel computing with the realization that clock speeds on CPU's have reached their limit. Heterogeneous Parallel Computing offers a solution to this problem by offloading computation that can be massively parallelized on the GPU's and running serial computation on the CPUs. Thus GP-GPU applications are widely used to solve several computational problems. For example, BarraCUDA is a software that uses GPU's to speed up the alignment of short sequence reads to a particular location on some reference genome.[2] GPU's have always been extensively used in the field of computer vision for image processing. It is to be noted, that most images are represented as dense matrices, and image processing algorithms render themselves to be effectively parallelized as they tend to operate on stencils(a fixed template of pixels). Since GPU's are particularly suited for vector computations, image processing applications typically are designed to utilize the massive parallelism offered by the GPU's.

In contrast in the field of Natural Language Processing(NLP), the models are typically sparse. This implies that language models have a sparse representation. Since the model is sparse, the computation does not admit to being effectively accomplished on the GPU. However recent advances in machine learning and natural language processing have resulted in the development of new language models that learn representations. These representations which are also termed *embeddings* are a dense representation. One popular technique of learning word representations is to use Deep Belief Networks. Deep

Belief Networks(DBNs) are neural networks which tend to have a large number of hidden layers. Training language models using DBNs requires massive amounts of training examples and thus requires massive computation. In this work, we investigate the performance of training such a language model on the GPU. At a high level, we measure the performance of the training in terms of the number of training examples processed per second. We evaluate the performance in terms of the speed-up achieved on the GPU and compare it against the CPU. We next briefly describe our experimental setup and describe our research methodology.

2.2 Overview of the Experimental Setup

We run all our experiments on the GPU on GEForce GT 570 . This GPU has 480 cores, with a processor clock speed of 1464MHz and a memory clock speed of 1900MHz. GPU's in the GEForce family are particularly suited for research applications. To train the language model, we use a well-known library called *Theano*[http://deeplearning.net/software/theano/]. Theano is a symbolic computation library that is particularly suited for machine learning applications. It allows for developers to succinctly specify how the parameters of the model are to be calculated. Using Theano to learn a model, allows the developer to implement a machine learning algorithm easily as the complexity involved in explicitly calculating the gradients of the loss function is abstracted out by Theano. Theano has functionality built in to calculate gradients and perform mathematical computation easily. Theano also transparently offloads computation on to the GPU if required. Since training a model, significantly involves computation required to learn a set of parameters that minimize a loss function, it is imperative that the functions implemented in Theano are efficient. This involves performing both macro level optimizations in optimizing the computational graph and micro level optimizations that target specific functions. Having described our experimental setup, we now describe at a high level, our research methodology in identifying bottle necks and optimizing these bottle necks.

2.3 Research methodology

We follow the below standard procedure to analyze and evaluate the performance bottlenecks:

1. Note the number of training examples processed per second on the CPU and GPU to establish a baseline.

2. Use a profiler to identify top hot spots.
3. Focus on the top hot spots and investigate how they can be optimized
    (a) Identify any computational graph optimizations possible to reduce computation
    (b) Optimize function calls by investigating how parallelism can be boosted.
    (c) Micro-optimize function calls
    (d) Unit Test the changes to ensure correctness
4. Repeat Steps 1 to 3 if required.

2.4 Experimental Results

2.4.1 Baseline

In this section, we present the baseline numbers for the GPU and the CPU. On the CPU, the mean training rate was 5512.6 examples/second($\sigma = 30.315$). On the GPU, the mean training rate was 1265.8 examples/second($\sigma = 20.604$).

The goal is to investigate the performance on the GPU and evaluate the bottle-necks involved. On scanning the logs, we note that the percentage of total time approximately spent on Theano processing was 96%.

This implies that it would be fruitful to analyze and focus on optimizations in Theano to help boost performance. In the next section, we analyze the performance of Theano using the built in Theano Profiler.

2.4.2 Profiling Theano

We profiled Theano to get some insight into what the performance hot spots are, so that our efforts could be channeled into optimizing those hot spots. We show the time taken per call and the fraction of time spent for these functions in the Table 1 .We note that there is 1 major hot spot, namely *GpuAdvancedIncSubTensor1*.

We thus narrow down our goal to optimizing the function *GpuAdvancedIncSubTensor1*. This function typically performs an operation called *advanced-indexing* which is an operation that is typically performed while calculating the gradient of parameter vector and updating them. In order to optimize this, it is crucial to understand what the operation of *advanced-indexing* does.

Table 2.1: Top 3 Hot spots in Theano

Theano Function	Fraction of time spent	Time per call to function
GpuAdvancedIncSubtensor1	81.7%	4.60×10^{-03}s
GpuElemwise	9.2%	6.93×10^{-05}s
GpuAlloc	1.7%	1.91×10^{-04}s

This operation operates takes as input 3 parameters: W and Y which are matrices and a vector I. The vector I refers to (indexes) rows in W. Given a row of W indexed by I, this operation adds the corresponding row of Y to it to form the output. This is done for each row indexed by I.

Having understood the operation that we are targeting to optimize, we now outline in the next section, the optimizations we investigated and present how the performance of this function improves with these optimizations.

2.4.3 Optimizing advanced indexing

To enable quick testing of advanced indexing, we wrote a standalone script that invokes the advanced indexing operation. This enables us to quickly test our fixes and ensure their correctness.

On examining the code for the advanced indexing, the following was noted:

1. The implementation of advanced indexing was done in Python and can be slower than an implementation in C. Hence it would be beneficial to rewrite the implementation in C to boost performance.

2. The code was not highly parallelized and had a low degree of parallelism. So we decided to not only write an implementation of advanced indexing in C, but parallelize it as well. This was done by writing a CUDA kernel which would perform the advanced indexing in parallel. More specifically instead of indexing each row sequentially, each row is indexed in parallel, and for each row, each cell in the row is added in parallel. This greatly boosts the parallelism of the algorithm.

3. Another micro-optimization that was tried was to also make the operation in-place at the cost of an extra memory copy. However this was shown to give diminishing benefits.

With these optimizations, the mean time taken for indexing 1000 rows is 3.6612 seconds($\sigma = 0.14116$) compared to the baseline where the mean time for indexing 1000 rows was 207.59 seconds ($\sigma = 2.9652$). We also noted specifically that the time per function call to advanced indexing had reduced by a factor of 50.

The above speedup thus results in Advanced Indexing no longer being the bottle-neck. In the training of the model, we do not index as many rows(as the context size and batch sizes are smaller) and hence the speed up in training is expected to be lower. One of the reasons for having small batch sizes is that the model converges faster. We in fact noted that increasing that batch size to 500 results in a much slower convergence rate of the model(as we will present in Section 2.4.6).

2.4.4 Speed up in rate of training

On making the above optimization, the mean rate of training is now 3742 examples/s($\sigma = 32.6496$) .

We thus note that we have achieved a reasonable speed up of $3 - 4$ times on the GPU with our optimizations and the performance on the GPU is comparable to that on the CPU. We also note that advanced indexing is no longer a major bottleneck in the training task. On performing these optimizations, it is imperative to do a further analysis on whether further optimizations are worth the "bang for the buck". This requires us to analyze exactly what is limiting the speedup. Thus in the next section we present our analysis on what is limiting the speed up on the GPU and highlight our findings below and outline the next steps.

2.4.5 Analysis of limits on training performance on GPU

We profiled the entire application(after we included all the optimizations above) and analyzed the profile log using NVIDIA Profiler(nvprof). From the profile, we extracted the following metrics:

1. *Compute Utilization:* This is the fraction of total time spent executing on the GPU. We would like the compute utilization to be high. If this ratio is small, this indicates that most of the time, the GPU's are idle. For our task, we note that the compute utilization is 7.4% which is low

2. *Compute to Memory Op Ratio:*Out of the time spent executing on the GPU, what fraction of time was spent doing computation to amount of time spent in transferring data to and from device is the Compute to Op Ratio. This ratio should be high and at least $10 : 1$. We noted that this metric is 66.72 which is high.

3. We also note the top 2 kernels as follows:

(a) *Composite Kernel*:This kernel performs an element wise operation on a C array which is contiguous in memory and is highly optimized with less scope for optimization.

(b) *copy_kernel*: This is a kernel which is a part of the BLAS Library.

(c) Also the above kernels are not expensive as they don't have expensive operations like exponentiation etc and hence are not bottlenecks.

Based on the above we conclude the following:

The performance of training on GPU is limited by the low compute utilization which implies that the GPU cores are mostly idle. The low compute utilization is a fallout of the model, as we are unable to fully utilize the GPU with our current implementation. One technique of improving compute utilization in this task would be to increase the batch size of examples which implies that each batch would contain more examples and thus increase the computation on the GPU. In the next section, we present our results on this task for increasing batch sizes.

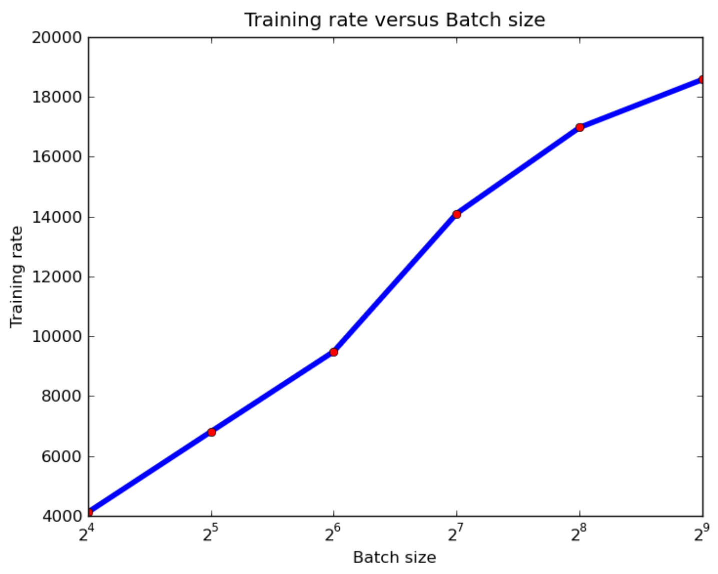
2.4.6 Effect of Batch Size on training and convergence rates

Currently, the batch size is set to 16. We decided to try a range of increasing batch sizes from 16 to 512 and measured the training rate and the time taken by the model to converge to an error less than 0.05. We make the following observations(see Figure 2.1):

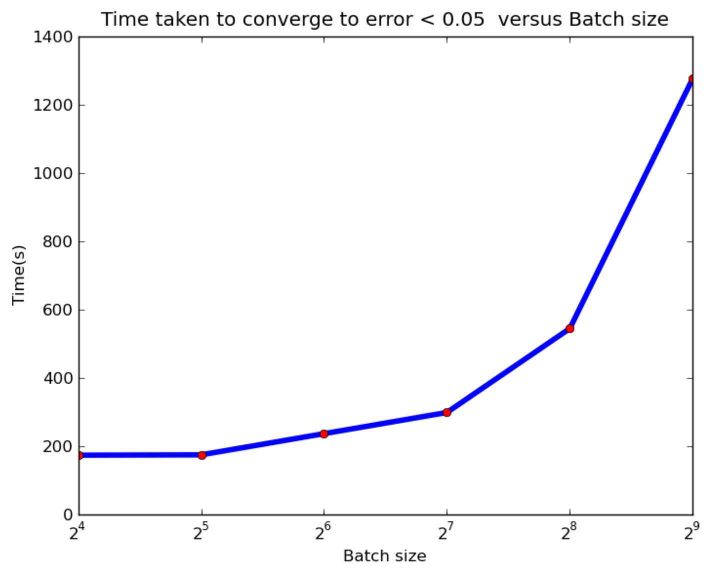
1. The training rate does increase as we increase the batch size.

2. The time taken to converge to a given error grows linearly (note the log scale on the X-axis) as we increase the batch size

The observation that the convergence rate decreases as we increase the batch size can be explained by the observations made by [3] that batch training is generally inefficient as compared to online training. We note that by increasing the batch size, the updates to weights accumulate and result in larger updates. This results in the gradient descent taking unreasonably large steps thus possibly overshooting the local minima on the error surface. We thus conclude by noting that increasing the batch size is not an effective strategy to speed up training as the model converges more slowly. In the next section, we highlight future research directions that could be investigated to speed up training.



(a) Effect of batch size on training rate.



(b) Effect of batch size on convergence rate

Figure 2.1: Effects of batch size on training and convergence rates.

2.5 Future Work

One area which could be investigated is to use the distributed algorithms for calculating gradients(using gradient descent) outlined by Jeffrey Dean et.al in

[4] in the model and evaluate its performance. These algorithms update the weight vector in a distributed fashion (with updates not being synchronized). It is claimed that distributed stochastic descent performs reasonably well for the models learnt. It would also be interesting to investigate if performance of training other models can be optimized on the GPU.A second direction would be to evaluate if we could effectively use GPU's for other pre-processing tasks like annotating the Web etc.

2.6 Conclusion

In this section, we briefly summarize our contributions below:

1. We were able to significantly optimize the operation of advanced indexing on the GPU in Theano.

2. We showed that this boosted the performance of training on the GPU by factor of 3-4 times and is comparable to the performance on CPU

3. We were able to analyze exactly what the limiting factors for training performance were on the GPU and showed that this performance was limited by low compute utilization.

4. We analyzed the effect of increasing batch size on the training speed and the convergence rate and concluded that the model converges slower for larger batch sizes even though we obtain a good speedup in training.

5. We also contributed back our optimizations to the Open source community so that these optimizations would benefit other members of the research community as well.

Chapter 3

Inducing Language Networks from Continuous Space Word Representations[1]

3.1 Introduction

Unsupervised feature learning (*deep learning*) utilizes huge amounts of raw data to learn representations that model knowledge structure and disentangle the explanatory factors behind observed events. Under this framework, symbolic sparse data is represented by lower-dimensional continuous spaces. Integrating knowledge in this format is the secret behind many recent breakthroughs in machine learning based applications such as speech recognition, computer vision, and natural language processing (NLP) [6].

We focus here on word representations (*word embeddings*) where each word representation consists of a dense, real-valued vector. During the pre-training stage, the representations acquire the desirable property that similar words have lower distance to each other than to unrelated words [7]. This allows the representations to utilize the abundance of raw text available to learn features and knowledge that is essential for supervised learning applications

[1]This is joint work with Bryan Perozzi and Rami Al Rafou' published as: Bryan Perozzi, Rami Al-Rfou, Vivek Kulkarni, and Steven Skiena. Inducing language networks from continuous space word representations. In Pierluigi Contucci, Ronaldo Menezes, Andrea Omicini, and Julia Poncela-Casasnovas, editors, *Complex Networks V*, volume 549 of *Studies in Computational Intelligence*, pages 261–273. Springer International Publishing, 2014. ISBN 978-3-319-05400-1. doi: 10.1007/978-3-319-05401-8_25. URL http://dx.doi.org/10.1007/978-3-319-05401-8_25.Original copyright notice is given to the publication in which the material was originally published. *With kind permission of Springer Science and Business media*

such as part-of-speech tagging, named entity recognition, machine translation, language modeling, sentiment analysis etc [8–11].

Several methods and algorithms have been proposed to learn word representations along different benchmarks for evaluation [12]. However, these evaluations are hard to comprehend as they squash the analysis of the representation's quality into abstract numbers. To enable better understanding of the actual structure of word relationships which have been captured, we have to address the problems that come with analyzing high-dimensional spaces (typically between 50-1000 dimensions). We believe that network induction and graph analysis are appropriate tools to give us new insights.

In this work, we seek to induce meaningful graphs from these continuous space language models. Specifically, our contributions include:

- **Analysis of Language Network Induction** - We propose two criteria to induce networks out of continuous embeddings. For both methods, we study and analyze the characteristics of the induced networks. Moreover, the networks generated lead to easy to understand visualizations.

- **Comparison Between Word Representation Methods** - We evaluate the quality of two well known words embeddings. We contrast between their characteristics using the analysis developed earlier.

The remainder of this paper is set up as follows. First, in Section 3.2, we describe continuous space language models that we consider. In Section 3.3, we discuss the choices involved with inducing a network from these embeddings and examine the resulting networks. Finally, we finish with a discussion of future work and our conclusions.

3.2 Continuous Space Language Models

The goal of a language model is to assign a probability for any given sequence of words estimating the likelihood of observing such a sequence. The training objective usually maximizes the joint probability of the training corpus. A continuous space probabilistic language model aims to estimate such probability distribution by, first, learning continuous representations for the words and phrases observed in the language. Such mapping is useful to cope with the curse of dimensionality in cases where data distribution is sparse as natural language. Moreover, these representations could be used as features for natural language processing applications, domain adaptation and learning transfer scenarios that involve text or speech.

More precisely, given a sequence of words $S = [w_1 \dots w_k]$, we want to maximize $P(w_1, \dots, w_k)$ and learn representations for words. During the training process the continuous space language model learns a mapping of words to points in $\mathbb{R}^d$, where d usually ranges between $20 - 200$. Prior to training we build a vocabulary V that consists of the most frequent $|V|$ words, we map each word to a unique identifier that indexes an embeddings matrix C that has a size of $|V| \times d$. The sequence S is now represented by a matrix $\left[C[w_1]^T \quad \ldots \quad C[w_k]^T\right]^T$, enabling us to compose a new representation of the sequence using one of several compositional functions. The simplest is to concatenate all the rows in a bigger vector with size kd. Another option is to sum the matrix row-wise to produce a smaller representation of size d. While the first respects the order of the words, it is more expensive to compute.

Given a specific sequence representation as an input, we will define a task that the model should solve, given the sequence representation as the only input. Our choice of the task ranges from predicting the next/previous word(s) to distinguishing between observed phrases and other corrupted copies of them. The chosen task and/or the compositional function influence the learned representations greatly as we will discuss later.

We will focus our investigations, here, on two embeddings which are trained with different tasks and compositional functions; the Polyglot and SkipGram embeddings.

3.2.1 Polyglot

The Polyglot project offers word representations for each language in Wikipedia [13]. For large enough Wikipedias, the vocabulary consists of the most frequent 100,000 words. The representations are learned through a procedure similar to the one proposed by [8]. For a given sequence of words $S_t = [w_{t-k} \ldots w_t \ldots w_{t+k}]$ observed in the corpus T, a corrupted sequence S'_t will be constructed by replacing the word in the middle w_t with a word w_j chosen randomly from the vocabulary V. Once the vectors are retrieved, we compose the sequence representation by concatenating the vectors into one vector called the projection layer S_t. The model is penalized through the hinge loss function,

$$\frac{1}{T}\sum_{t=1}^{t=T} |1 - score(S'_t) + score(S_t)|_+$$

where *score* is calculated through a hidden layer neural network

$$score(S_t) = W_2(tanh(W_1 S_t + b_1)) + b_2.$$

For this work, we use the Polyglot English embeddings[2] which consist of the 100,000 most frequent words in the English Wikipedia, each represented by a vector in $\mathbb{R}^{64}$.

3.2.2 SkipGram

While the Polyglot embeddings consider the order of words to build the representation of any sequence of words, the SkipGram model proposed by [14] maximizes the average log probability of the context words independent of their order

$$\frac{1}{T}\sum_{t=1}^{T}\left[\sum_{j=-k}^{k}\log p(w_{t+j}|w_t)\right]$$

where k is the size of the training window. This allows the model to scale to larger context windows. In our case, we train a SkipGram model[3] on the English Wikipedia corpus offered by the Polyglot project for the most frequent 350,000 words with context size k set to 5 and the embeddings vector size set to 64.

3.2.3 Random

In order to have a baseline, we also generate random embeddings for the most frequent 100,000 words. The initial position of words in the Polyglot embeddings were sampled from a uniform distribution, therefore, we generate the random embedding vectors by sampling from $\mathcal{U}(\bar{m}-\sigma, \bar{m}+\sigma)$, where $\bar{m}$ and σ are the mean and standard deviation of the trained Polyglot embeddings' values respectively. This baseline allows us to see how the language networks we construct differ from networks induced from randomly initialized points.

3.3 Word Embedding Networks

We now consider the problem of constructing a meaningful network given a continuous space language model. As there are a variety of ways in which such a network could be induced, we start by developing a list of desirable properties for a language network. Specifically, we are seeking to build a network which:

[2]Polyglot embeddings and corpus available at `http://bit.ly/embeddings`
[3]SkipGram training tool available at `https://code.google.com/p/word2vec/`

1. **Is Connected** - In a connected graph, all the words can be related to each other. This allows for a consistent approach when trying to use the network to solve real-world problems.
2. **Has Low Noise** - Minimizing the spurious correlations captured by our discrete representation will make it more useful for application tasks.
3. **Has Understandable Clusters** - We desire that the community structure in the network reflects the syntactic and semantic information encoded in the word embeddings.

We also require a method to compute the distance in the embedding space. While there are a variety of metrics that could be used, we found that Euclidean distance worked well. So we use:

$$dist(x,y) = ||x-y||_2^2 = (\sum_{i=1}^{m}(x_i - y_i)^2)^{(1/2)} \tag{3.1}$$

where x and y are words in an d-dimensional embedding space ($x, y \in \mathbb{R}^d$). With these criteria and a distance function in hand, we are ready to proceed. We examine two approaches for constructing graphs from word embeddings, both of which seek to link words together which are close in the embedding space. For each method, we induce networks for the $20,000$ most frequent words for each embedding type, and compare their properties.

3.3.1 k-Nearest Neighbors

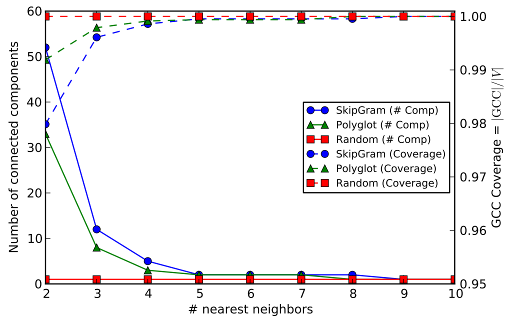
The first approach we will consider is to link each word to the k closest points in the embedding space. More formally, we induce a set of directed edges through this method:

$$E_{knn} = \{(u,v) : \min_x dist(u,v)\} \quad \forall u,v \in V, x \leq k \tag{3.2}$$

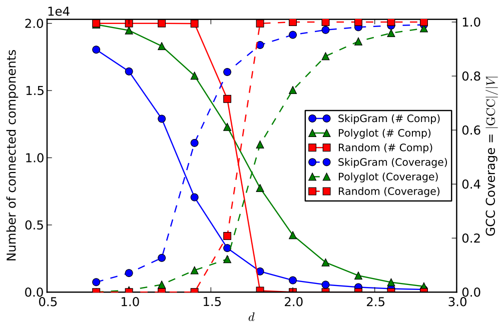
where $\min_x$ denotes the rank of the x-th number in ascending sorted order (e.g. $\min_0$ is the minimum element, $\min_1$ the next smallest number). After obtaining a directed graph in this fashion, we convert it to an undirected one.

The resulting undirected graph does not have a constant degree distribution. This is due to the fact that the nearest-neighbor relation may not be symmetric. Although all vertices in the original directed graph have an out-degree of k, their orientation in the embedding space means that some vertices will have higher in-degrees than others.

Results from our investigation of basic network properties of the k-NN embedding graphs are shown in Figures 3.1 and 3.2. In (3.1a) we find that the



(a) k-NN Coverage



(b) d-Threshold Coverage

Figure 3.1: **Graph Coverage**. The connected components and relative size of the Giant Connected Component (GCC) in graphs created by both methods. We see that very low values of k quickly connect the entire network (3.1a), while values of d appear to have a transition point before a GCC emerges (3.1b).

embedding graphs have few disconnected components, even for small values of k. In addition, there is an obvious GCC which quickly emerges. In this way, the embeddings are similar to the network induced on random points (which is fully connected at $k = 2$). We performed an investigation of the smaller connected components when k was small, and found them to contain dense groupings of words with very similar usage characteristics (including ordinal values, such as Roman numerals (`II,III,IV`)).

In (3.2a) we see that the clustering coefficient initially grows quickly as we add edges to our network ($k \leq 6$), but has leveled off by ($k = 20$). This tendency to bridge new clusters together, rather than just expand existing ones, may be related to the *instability* of the nearest neighbor [15] in high dimensional spaces. In (3.2b), we see that the networks induced by the k-NN are not only connected, but have a highly modular community structure.

3.3.2 d-Proximity

The second approach we will consider is to link each word to all those within a fixed distance d of it:

$$E_{proximity} = \{(u, v) : dist(u, v) < d\} \quad \forall u, v \in V \tag{3.3}$$

We perform a similar investigation of the network properties of embedding graphs constructed with the d-Proximity method. The results are shown in Figures 3.1 and 3.2. We find that networks induced through this method quickly connect words that are near each other in the embedding space, but do not bridge distant groups together. They have a large number of connected components, and connecting 90% of the vertices requires using a relatively large value of d (3.1b).

The number of connected components is closely related to the average distance between points in the embedding space (around $d =$(3.25, 3.80, 2.28) for (SkipGram, Polyglot, Random)). As the value of d grows closer to this average distance, the graph quickly approaches the complete graph.

Figure 3.2a shows that as we add more edges to the network, we add triangles at a fast rate than using the k-NN method.

3.3.3 Discussion

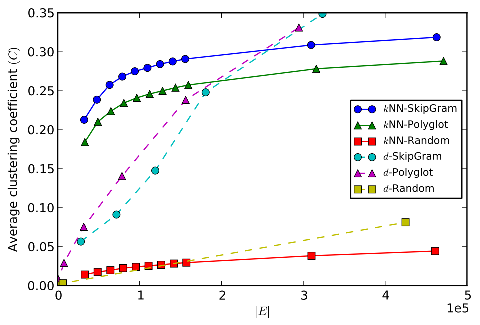
Here we discuss the differences exposed between the methods for inducing word embeddings, and the differences exposed between the embeddings themselves.

Comparison of Network Induction Methods.

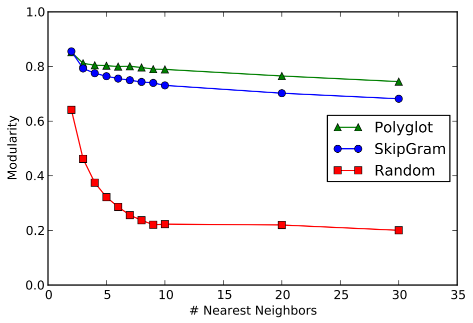
Which method then, provides the better networks from word embeddings? To answer this question, we will use the properties raised at the beginning of this section:

1. **Connectedness** - Networks induced through the k-NN method connect much faster (as a function of edges) than those induced through d-Proximity (Fig. 3.1). Specifically, the network induced for $k = 6$ has nearly full coverage (3.1a) with only 100K edges (3.2a).

2. **Spurious Edges** - We desire that our resulting networks should be modular. As such we would prefer to add edges between members of a community, instead of bridging communities together. For low values



(a) Clustering Coeff., C



(b) k-NN Modularity.,Q_{knn}

Figure 3.2: **Community Metrics**. In (3.2a), C shown for k = [2,30] and d = [0.8,1.6] against number of edges in the induced graph. When the total number of edges is low ($|E| < 150,000$), networks induced through the k-NN method have more closed triangles than those created through d-Proximity. In (3.2b), Q_{knn} starts high, but slowly drops as larger values of k include more spurious edges.

of $|E|$, the k-NN approach creates networks which have more closed triangles (3.2a). However this does not hold in networks with more edges.

3. **Understandable Clusters** - In order to qualitatively examine the quality of such a language network, we induced a subgraph with the k-NN of the most frequent 5,000 words in the Polyglot embeddings for English. Figure 3.3 presents the language network constructed for ($k = 6$).

According to our three criteria, k-NN seems better than d-Proximity. In addition to the reasons we already listed, we prefer k-NN as it seems to require less parameterization (d-Proximity has a different optimal d for each embedding type).

Comparison of Polyglot and SkipGram.

Having chosen to use k-NN as our preferred method for inducing language networks, we now examine the difference between the Polyglot and SkipGram networks.

Clustering Coefficient. We note that in Figure 3.2a, the SkipGram model has a consistently higher clustering coefficient than Polyglot in k-NN networks. A larger clustering coefficient denotes more triangles, and this may indicate that points in the SkipGram space form more cohesive local clusters than those in Polyglot. Tighter local clustering may explain some of the interesting regularities observed in the SkipGram embedding [16].

Modularity. In Figure 3.2b, we see that Polyglot modularity is consistently above the SkipGram modularity. SkipGram's embeddings capture more semantic information about the relations between words, and it may be that causes a less optimal community structure than Polygot whose embeddings are syntactically clustered.

Clustering Visulizations. In order to understand the differences between the language networks better, we conducted an examination of the clusters found using the Louvain method [17] for modularity maximization. Figure 3.4 examines communities from both Polyglot and SkipGram in detail.

3.4 Related Work

Here we discuss the relevant work in language networks, and word embeddings. There is also related work on the theoretical properties of nearest neighbor graphs, consult [19] for some basic investigations.

3.4.1 Language Networks

Word Co-occurrences. One branch of the study of language as networks seeks to build networks directly from a corpus of raw text. [20] examine word co-occurrence graphs as a method to analyze language. In their graph, edges connect words which appear below a fixed threshold ($d \leq 2$) from each other in sentences. They find that networks constructed in this manner show both small world structure, and a power law degree distribution. Language networks based on word co-occurrence have been used in a variety of natural language processing tasks, including motif analysis of semantics [21], text summarization [22] and resolving disambiguation of word usages [23].

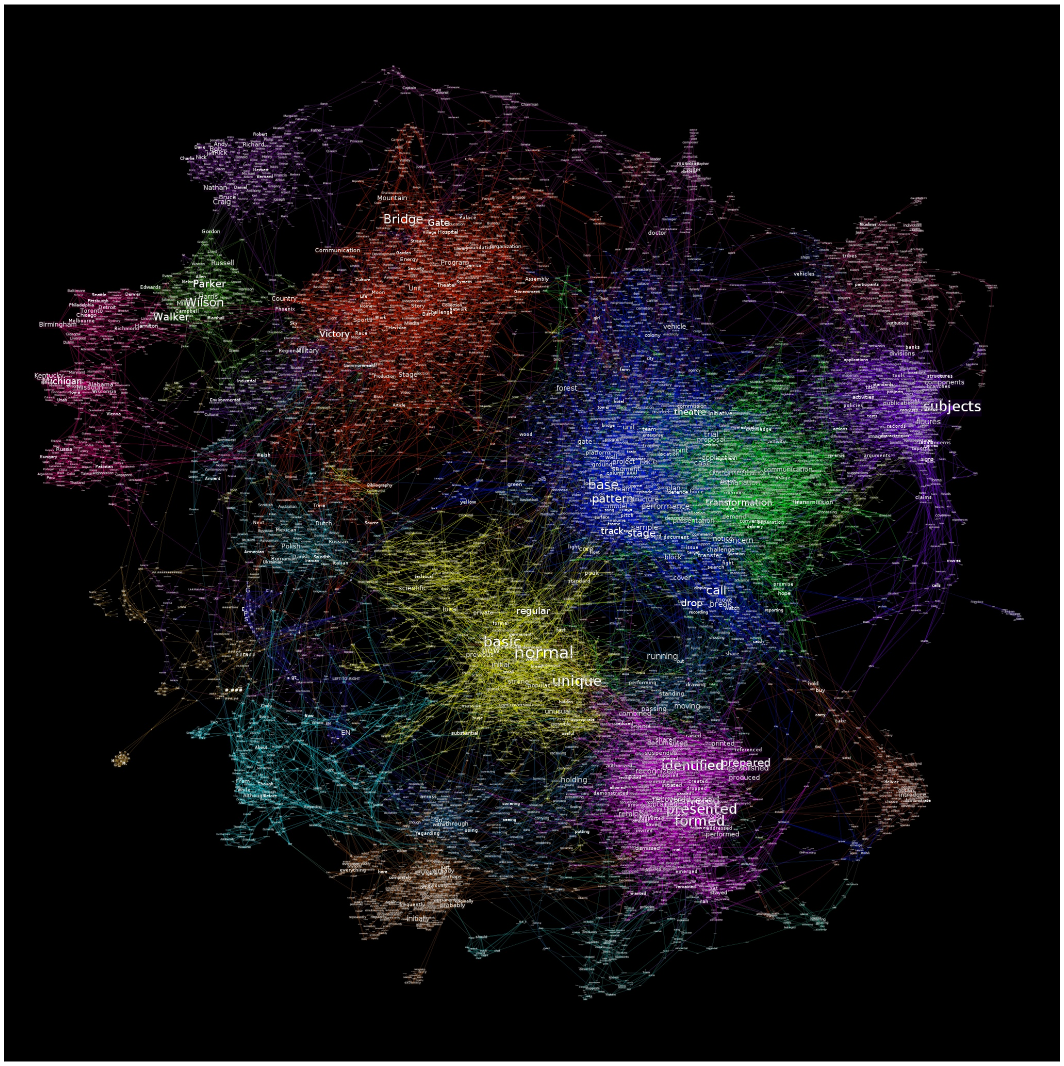
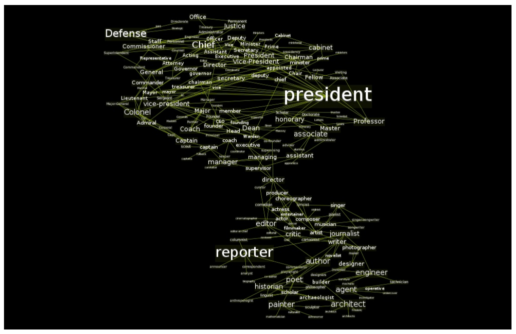
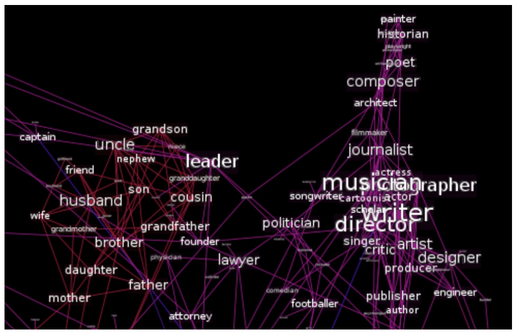


Figure 3.3: **Polyglot Nearest Neighbor Graph**. Here we connect the nearest neighbors ($k = 6$) of the top 5,000 most frequent words from the Polyglot English embeddings. Shown is the giant connected component of the resulting graph ($|V| = 11,239$; $|E| = 26,166$). Colors represent clusters found through the Louvain method (modularity $Q = 0.849$). Vertex label size is determined by its PageRank. Best viewed in color.

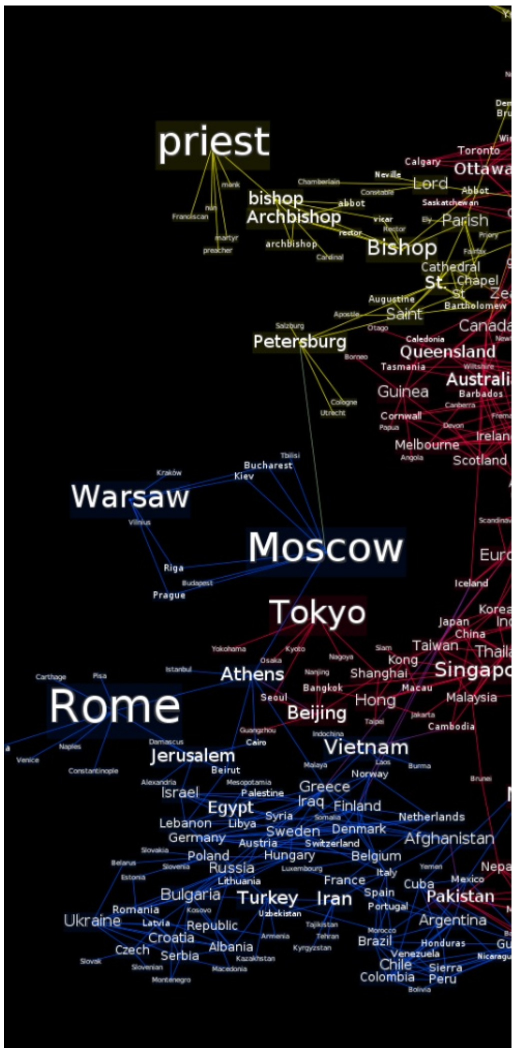
Hypernym relations. Another approach to studying language networks relies on studying the relationships between words exposed by a written language reference. [24] use a thesaurus to construct a network of synonyms, which they find to find to exhibit small world structure. In [25], Sigman and Cecchi investigate the graph structure of the Wordnet lexicon. They find that the



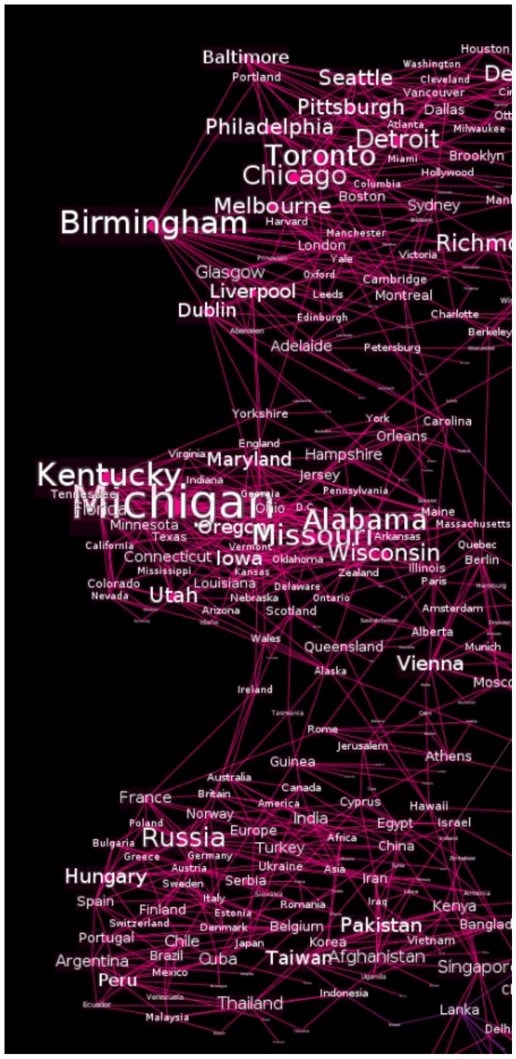
(a) Professions (SkipGram)



(b) Professions (Polyglot)



(c) Locations (SkipGram)



(d) Locations (Polyglot)

Figure 3.4: Comparison of clusters found in Polyglot and SkipGram language networks. Polyglot clusters viewed in context of the surrounding graph, SkipGram clusters have been isolated to aide in visualization. SkipGram's bag-of-words approach favors a more semantic meaning between words, which can make its clusters less understandable (Note how in Figure 3.4c `Petersburg` is included in a cluster of religious words, because of `Saint`.) Images created with Gephi [18].

semantic edges in Wordnet follow scale invariant behavior and that the inclusion of polysemous edges drastically raises the clustering coefficient, creating a small world effect in the network.

Relation to our work. Much of the previous work in language networks build networks that are prone to noise from spurious correlations in word co-occurrence or infrequent word senses [20, 25]. Dimensionality reduction techniques have been successful in mitigating the effects of noise in a variety of domains. The word embedding methods we examine are a form of dimensionality reduction that has improved performance on several NLP tasks and benchmarks.

The networks produced in our work are considerably different from language networks created by previous work that we are aware of. We find that our degree distribution does appear to follow a power-law (like [20, 24, 25]) and we have some small world properties like those present in those works (such as $C \gg C_{random}$). However, the average path length in our graphs is considerably larger than the average path length in random graphs with the same node and edge cardinalities. Table 3.1 shows a comparison of metrics from different approaches to creating language networks.[4]

	$\lvert V\rvert$	$\lvert E\rvert$	C	C_{random}	pl	pl_{random}	γ
[20](UWN)	$478,773$	1.77×10^7	0.687	1.55×10^{-4}	2.63*	3.03	-1.50,-2.70
[20](RWN)	$460,902$	1.61×10^7	0.437	1.55×10^{-4}	2.67*	3.06	-1.50,-2.70
[24](PRE)	$30,244$	–	0.53	0.002	3.16	–	–
Polyglot, 6-NN	$20,000$	$96,592$	0.241	0.0004	6.78*	4.62*	-1.31
SkipGram, 6-NN	$20,000$	$94,172$	0.275	0.0004	6.57*	4.62*	-1.32

Table 3.1: A comparison of properties of language networks from the literature against those induced on the 20,000 most frequent words in the Polyglot and SkipGram Embeddings. (C clustering coefficient, pl average path length, γ exponent of power law fits to the degree distribution) '*' denotes values which have been estimated on a random subset of the vertices.

3.4.2 Word Embeddings

Distributed representations were first proposed by [26], to learn a mapping of symbolic data to continuous space. These representations are able to capture fine grain structures and regularities in the data [16]. However, training these

[4]Our induced networks available at `http://bit.ly/inducing_language_networks`

models is slow due to their complexity. Usually, these models are trained using back-propagation algorithm [27] which requires large amount of computational resources. With the recent advancement in hardware performance, [28] used the distributed representations to produce a state-of-the-art probabilistic language model. The model maps each word in a predefined vocabulary V to a point in $\mathbb{R}^d$ space (word embeddings). The model was trained on a cluster of machines for days. More applications followed, [8] developed SENNA, a system that offers part of speech tagger, chunker, named entity recognizer, semantic role labeler and discriminative syntactic parser using the distributed word representations. To speed up the training procedure, importance sampling [29] and hierarchical softmax models [30, 31] were proposed to reduce the computational costs. The training of word representations involves minimal amount of language specific knowledge and expertise. [13] trained word embeddings for more than a hundred languages and showed that the representations help building multilingual applications with minimal human effort. Recently, SkipGram and Continuous bag of words models were proposed by [14] as simpler and faster alternatives to neural network based models.

3.5 Conclusions

We have investigated the properties of recently proposed distributed word representations, which have shown results in several machine learning applications. Despite their usefulness, understanding the mechanisms which afford them their characteristics is still a hard problem.

In this work, we presented an approach for viewing word embeddings as a language network. We examined the characteristics of the induced networks, and their community structure. Using this analysis, we were able to develop a procedure which develops a connected graph with meaningful clusters. We believe that this work will set the stage for advances in both NLP techniques which utilize distributed word representations, and in understanding the properties of the machine learning processes which generate them.

Much remains to be done. In the future we would like to focus on comparing word embeddings to other well known distributional representation techniques (e.g. LDA/LSA), examining the effects of different vocabulary types (e.g. topic words, entities) on the induced graphs, and the stability of the graph properties as a function of network size.

Acknowledgments

This research was partially supported by NSF Grants DBI-1060572 and IIS-1017181, with additional support from TASC Inc, and a Google Faculty Research Award.

Chapter 4

Multilingual Named Entity Recognition[1]

4.1 Introduction

Named entity recognition (NER) is an essential pre-processing stage in many NLP and Information Retrieval (IR) systems. Most of the successful approaches in NER rely on supervised learning [32, 33]. Despite several shared tasks (MUC, CoNLL, ACE), many languages still lack the necessary resources to build NER annotators. In addition, it is often hard to find NLP practitioners proficient in every language of interest. This makes developing multilingual applications quite tedious.

Here, we address both bottlenecks by developing a system which we call MMNER (Massive Multilingual NER). MMNER uses language-independent techniques which enable the automated construction of NER annotators for 40 languages, through the use of link structure and distributed word representations. Distributed word representations capture the semantic and syntactic characteristics of words through unsupervised learning. They have been used successfully as features in NLP applications [34, 35]. They have also been proposed as a cornerstone for developing multilingual applications [13].

To address the lack of human annotated datasets for multilingual NER, our method uses the internal links between the pages of WIKIPEDIA. When the links refer a page mentioning an entity, the sentence is considered a candidate for inclusion in our corpus. WIKIPEDIA covers well over a hundred languages, and its link structure has been shown to result in better generalization than using other human-annotated datasets [36].

Previous work has utilized the WIKIPEDIA link structure through linguisti-

[1]This is joint work with Bryan Perozzi and Rami Al Rafou'.

cally inspired, language-specific preprocessing rules [37–40]. In stark contrast, our method uses only *language independent* transformations to generate a training dataset.

Our major contributions are the following:

1. **NER annotators and datasets** for 40 languages.[2] These annotators are valuable, especially for resource scarce languages, like Serbian, Indonesian, Persian and Hebrew.

2. **Techniques** for dealing with missing labels in annotated datasets. We propose *oversampling* of entity labels, and *co-reference resolution* to deal with bias introduced by WIKIPEDIA style guidelines. These techniques improve the performance of our trained models by at least 45% F_1 on CoNLL datasets.

3. **Comparative analysis** using statistical machine translation to evaluate the performance of our system in resource-scarce languages.

Section 4.2 defines the problem and presents a term-based classification formulation. Section 4.3 highlights our approach to generate training data, and Section 4.4 evaluates the results. We finalize with a comparative study across all languages in Section 4.5.

4.2 Semi-supervised Learning

The Named Entity Recognition (NER) task is to identify sequences of tokens as entities, and classify them into one of several categories.

In order to deal with noisy training data, we make different design decisions. First, we use distributed word representations (or *word embeddings*) as fixed features to provide prior knowledge of language that is independent of the labeled data. Secondly, to minimize the effect of noise in inference, we use a discriminative classifier (a neural network). This avoids errors which can occur from propagating labels as features.

4.2.1 Learning Word Embeddings

Given a language with vocabulary V, a word embedding is a mapping function $\Phi\colon w \mapsto \mathbb{R}^d$, where $w \in V$ and $d \in [50, 500]$. We use the Polyglot[3] embeddings [13] as our sole features for each language under investigation. The Polyglot

[2]Available at http://entityextractor.appspot.com.

[3]Available: http://bit.ly/embeddings

embeddings were trained using an objective function proposed by [41], on Wikipedia text without any labeled data. The objective function respects the order of words, which makes it easier to distinguish proper nouns and capitalized words without manual encoding or processing.

4.2.2 Discriminative Learning

We consider NER to be a classification problem. More formally, let $\mathcal{W}_i^n = (w_{i-n} \cdots w_i \cdots w_{i+n})$ be a phrase centered around the word w_i with a window of size n. We seek to learn a model $F\colon \mathcal{W}_i^n \mapsto \mathcal{Y}$, where $\mathcal{Y}$ is the set of tags. First, we map the phrase $\mathcal{W}_i^n$ to its embedding representation

$$\mathbf{\Phi}_i^n = [\Phi(w_{i-n}); \ldots; \Phi(w_i); \ldots; \Phi(w_{i+n})]$$

Next, we learn a model Ψ_y to score tag y given $\mathbf{\Phi}_i^n$, i.e $\Psi_y : \mathbf{\Phi}_i^n \mapsto \mathbb{R}$, using a neural network with one hidden layer of size h

$$\Psi_y(\mathbf{\Phi}_i^n) = \mathbf{s}^T(\tanh(\mathbf{W}\mathbf{\Phi}_i^n + \mathbf{b})) \tag{4.1}$$

where $\mathbf{W} \in \mathbb{R}^{h\times(2n+1)d}$ and $\mathbf{s} \in \mathbb{R}^h$ are the first and second layer weights of the neural network, and $\mathbf{b} \in \mathbb{R}^h$ are the bias units of the hidden layer. Finally, we construct a one-vs-all classifier F and penalize it by the following hinge loss,

$$J = \frac{1}{m}\sum_{i=1}^{m} max\Big(0,\ 1 - \Psi_{t_i}(\mathbf{\Phi}_i^n) + \max_{\substack{y\neq t_i\\ y\in\mathcal{Y}}} \Psi_y(\mathbf{\Phi}_i^n)\Big)$$

where t_i is the correct tag of the word w_i, and m is the size of the training set.

4.2.3 Optimization

We learn the parameters $\theta = (\mathbf{s}, \mathbf{W}, \mathbf{b})$ via backpropagation [42] with stochastic gradient descent [43]. As the stochastic optimization performance is dependent on a good choice of the learning rate, we automate the learning rate selection through an adaptive update procedure [44]. This results in separate learning rates η_i for each individual parameter, θ_i. More specifically the learning rate at step t for parameter i is given by the following:

$$\eta_i(t) = \frac{1.0}{\sqrt{\sum_{s=1}^{t}\left(\frac{\partial J(s)}{\partial \theta_i(s)}\right)^2}} \tag{4.2}$$

4.3 Extracting Entity Mentions

In this section, we outline our two-step approach for creating a named entity training corpus from Wikipedia. First, we categorize the pages into entity classes, using Freebase. Second, we extend the annotations through oversampling and co-reference resolution.

4.3.1 Article Categorization

Wikipediaconsists of interlinked articles on a variety of topics. We categorize the topics into the following categories, $\mathcal{Y} = \{$PERSON, LOCATION, ORGANIZATION, NONENTITY$\}$ using Freebase [45] attributes.

The result is a mapping of Wikipedia page titles and their redirects to $\mathcal{Y}$. If an internal link points to any of these titles, we consider it an entity mention. Table 4.1 shows the percentage of pages that are covered by Freebase for some of the languages we consider. The entity coverage varies with each language, and this greatly changes the label distribution of the generated training data.

4.3.2 Missing Links

Unfortunately, generating a training dataset directly from the link structure results in very poor performance with $\text{Dev}_{F_1} < 10\%$ in English, Spanish and Dutch. This is a consequence of Wikipedia style guidelines[4]. Editors are instructed to link the first mention in the page, but not later ones. This results in leaving most entity mentions unmarked. This effect is examined in Table 4.2, which contrasts the percentage of words that are covered by entity phrases (ρ) in both CoNLL and Wikipedia. In the reminder of this section we present our solution to this problem.

Oversampling

In the context of a cost sensitive learner, such a skewed label distribution leads to under-performing models [46, 47]. To overcome this label bias, we change the label distribution by oversampling from the entity classes. The intuition is that untagged words are not necessarily non-entities. Conversely, we have high confidence in the links which have been explictly tagged by users. Therefore, we consider all the non-entity words to be negatively labeled. We oversample from the positive class (the entity tags), uniformly.

[4]http://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Linking

Language	Coverage	Language	Coverage
Malay	37.8%	Arabic	15.6%
English	35.2%	Dutch	14.7%
Spanish	24.8%	Swedish	10.2%
Greek	24.3%	Hindi	8.8%

Table 4.1: Percentage of Wikipedia pages identified by Freebase as being about entities. There is a wide disparity across languages.

Co-reference Resolution

While oversampling mitigates the effect of the skewed label distributions, it does not address the stylistic bias with which Wikipedia editors create links. The first bias is to link only the first mention of a name in an article. This canonical mention is usually the *full* name of an entity, and not the abbreviated form used throughout the remainder of the article. Moreover, there are no self-referential links inside an entity's article (e.g. on `Barack Obama`'s page, none of the sentences mentioning him are linked).

In 200K examples tagged with person, we found 45K examples belong to three terms mentions, 140K to two terms mentions and only 15K belonging to single term mentions. This bias against single term mentions in links does not reflect the true distribution of named entities in the text.

In order to more accurately reflect the nature of the data, we extend our annotations differently for each page. If a word appeared in an entity mention, or in the title (for entity pages), then we consider all appearances of this word in the current page to be annotated with the same tag. In the case of multiple tags for the same word, we use the most frequent tag in the article.

For example: after this procedure, every mention of '`Barack Obama`', `Barack`, and `Obama` in the article on `Barack_Obama` will be considered a link referring to a person entity. In order to avoid mislabeling functional words which appear in links (e.g. `of`, `the`, `de`) we exclude the most frequent 1000 words in our vocabulary. This extension can be viewed as first-order coreference resolution stage where we match mentions depending on string matching.

4.4 Experiments

In this section, we evaluate the performance of mmner stage-wise on CoNLL datasets to demonstrate the efficiency of the proposed solutions to deal with missing links in the Wikipedia markup and missing attributes in Freebase.

Language	% Entity Words	
	CoNLL	Wiki
English	16.76%	2.34%
Spanish	12.60%	2.12%
Dutch	9.29%	2.28%

Table 4.2: Percentage of words that are covered by entities phrases in each corpus. Hyperlinks under-represent named entities in text mandating oversampling.

4.4.1 Testing Datasets

We evaluate our models on the CoNLL 2002 Spanish and Dutch datasets [48], and the CoNLL 2003 English dataset [49]. There are a variety of orthographic and contextual differences between Wikipedia and CoNLL. Training on Wikipediaresults in low scores on CoNLL testing datasets compared to models trained on CoNLL directly [36].

Common differences between the datasets include: trailing periods, leading delimiters, and modifiers and annotators' disagreements. Specifically, the English CoNLL dataset has an over representation of upper case words and sports teams. In the Dutch dataset, country names were abbreviated after the journalist name. For example, *Spain* will be mapped to *Spa* and *Italy* to *Ita*. This leads to more out of vocabulary (OOV) terms for which Polyglot does not have word embeddings. Such notational differences pose more harm to our performance than other approaches because we do not rely on any tailored preprocessing steps.

4.4.2 Results

Here we present the experiment results of mmner evaluated on CoNLL. We train our annotators on datasets of size 100K examples for 50 epochs.

Oversampling (mmner_{OS}): Figure 4.1 shows the results of oversampling. The first point corresponds to the original distribution of labels in Wikipedia text, where $\rho \cong 2.5\%$. We observe that regardless of the chosen ρ, oversampling improves the results. This improvement is quite stable when $0.25 < \rho < 0.75$, and the Exact F_1 score is increased by at least 40% for all languages we consider.

Coreference ($\text{mmner}_{\text{COR+OS}}$): We apply coreference resolution stage on Wikipedia text before oversampling. Figure 4.2 shows that the F_1 improvements we observe in English, Spanish, and Dutch are significant, especially when $\rho \leq 0.5$. Most of this improvement is due to higher recall on the tag person.

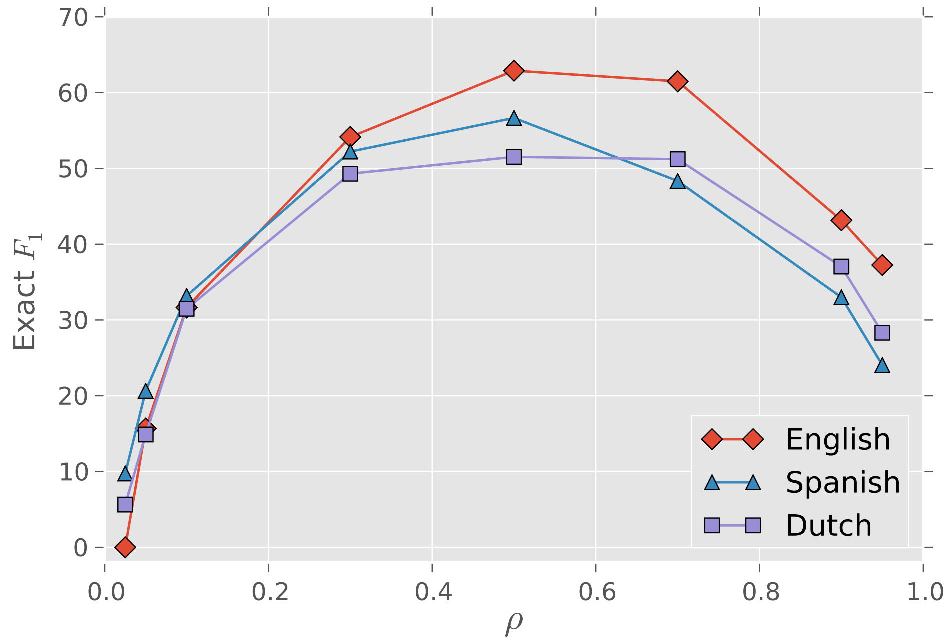


Figure 4.1: Performance of oversampling entities (MMNER$_{\text{OS}}$) on DEV datasets.

Table 4.3 shows that oversampling (MMNER$_{\text{OS}}$) alone is able to get competitive results. With coreference applied to the data first (MMNER$_{\text{COR+OS}}$), we outperform previous work on English and Spanish. Most of MMNER Dutch errors appear in the category of ORGANIZATION. MMNER shows this performance without applying any language-specific rules.

4.5 Comparative Study

In order to evaluate MMNER performance on the languages we do not have human annotated data for, in this case 37 languages, we develop a benchmark based on statistical machine translation. 1) We annotate English WIKIPEDIA sentences using STANFORD NER. 2) We randomly pick 1500 sentences that have at least one entity detected. 3) We translate these sentences using GOOGLE translate to 40 languages. 4) We compare the number of entity chunks our annotators find to the ones detected by STANFORD NER.

Specifically, we define the count of an entity type e appearing in an annotated English sentence S_{en} to be $C_e(S_{en})$, and the count that our annotator produces in the destination language L to be $C_e(S_L)$. We define two classes of

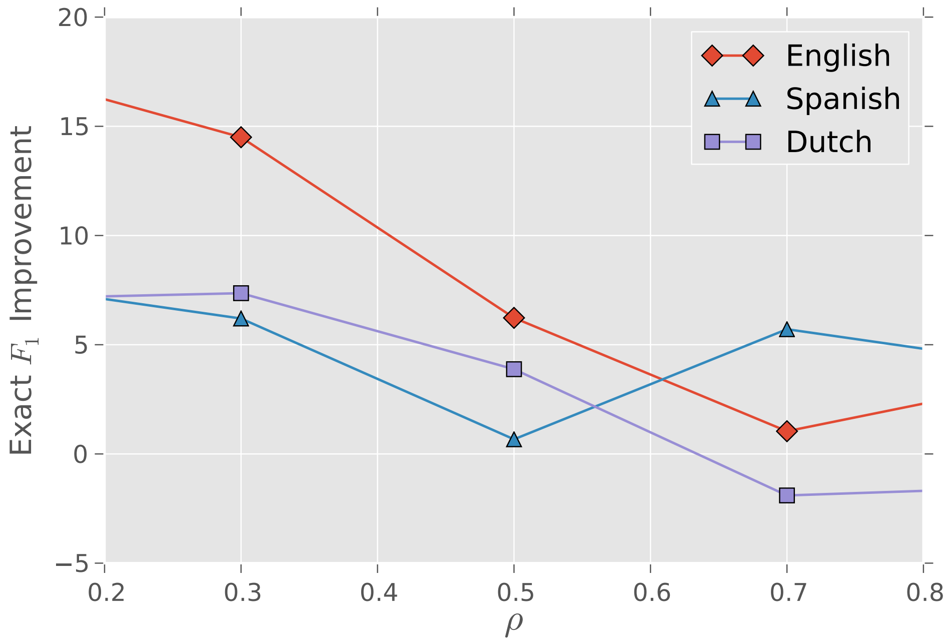


Figure 4.2: Performance improvement on CoNLL Dev datasets after applying coreference stage first on Wikipedia before oversampling.

error measures: omitting entities $\mathcal{E}_{\mathcal{M}}$ and adding entities $\mathcal{E}_{\mathcal{A}}$, as the following

$$Z_e = \sum_S C_e(S_{en})$$
$$\mathcal{E}_{\mathcal{M}}(e, L) = \sum_S \frac{max(0, C_e(S_{en}) - C_e(S_L))}{Z_e}$$
$$\mathcal{E}_{\mathcal{A}}(e, L) = \sum_S \frac{max(0, C_e(S_L) - C_e(S_{en}))}{Z_e}$$

For brevity, we show our error analysis only for the person category. Similar conclusion could be drawn for other categories. Figure 4.3 shows the performance of our system compared to other annotators; OpenNLP {English, Spanish, Dutch}, NLTK English, and Stanford German. Languages with the largest number of Wikipedia pages like English (`en`), French (`fr`), Spanish (`es`) and Portuguese (`pt`) show strong and consistent performance.

Our benchmark also highlights language specific issues. The poor performance in Japanese (`ja`) is due to the mismatch between the embedding vocabulary and the evaluation tokenizer. Vietnamese (`vi`) annotator is aggressive

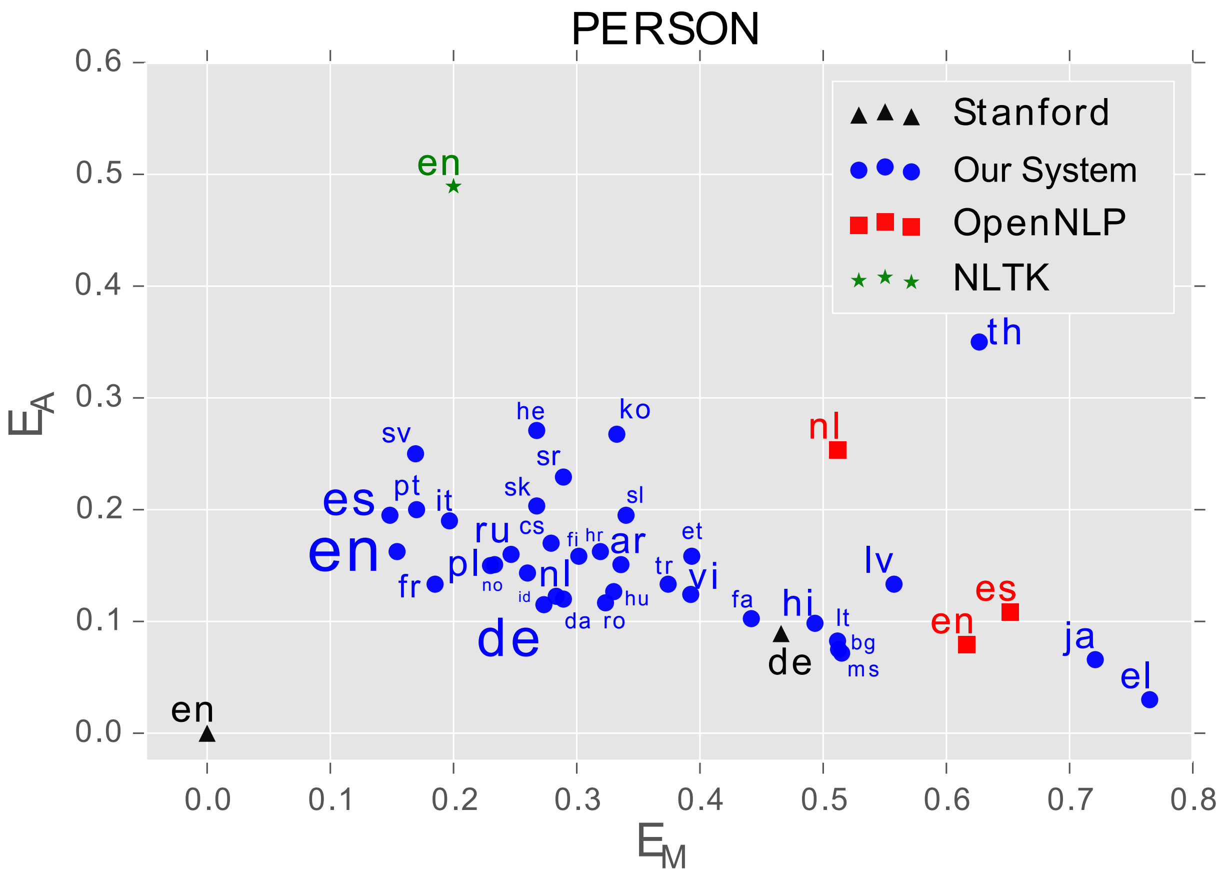


Figure 4.3: Error analysis of MMNER on PERSON category compared to other systems.

DEV	English	Spanish	Dutch
MMNER$_{OS}$	62.9	56.7	53.2
MMNER$_{COR+OS}$	**72.0**	59.0	56.9
(Nothman 2013)	67.9	**60.7**	**62.2**
TEST			
MMNER$_{OS}$	58.5	58.5	51.5
MMNER$_{COR+OS}$	**66.9**	**61.5**	56.3
(Nothman 2013)	61.3	61.0	**64.0**

Table 4.3: Exact F_1 Performance on CoNLL corpora. The best ρ on the DEV is used to set ρ for the TEST dataset.

in annotating chunks as LOCATION ($\mathcal{E}_{\mathcal{A}} = 0.6$) because there are more pages Vietnamese FREEBASE identified as LOCATION. Google Translate does not translate the entities efficiently in some languages. This is the reason behind the bad performance of Greek (`el`) and Thai (`th`). We outperform OPENNLP and NLTK, by a significant margin. MMNER German annotator covers more PERSON entities than STANFORD without adding many false positives.

4.6 Related Work

WIKIPEDIA has been used for generating NER datasets, [37] and [38] used first sentences of WIKIPEDIA articles to build their datasets. The first used Wordnet as a supporting resource while the second relied on the existence of a part of speech tagger to clean their annotations. Following their efforts on utilizing WIKIPEDIA metadata, [50] utilized the inter-wiki links to build annotated dataset using several hand crafted rules. Such approach has been extended to several languages [39, 40]. Parallel data was used to help building multilingual NER [51]. We distinguish our work by avoiding any language specific knowledge or expertise. We use automatically learned features and process the noisy datasets with no lexical, morphological or syntactic rules. This simplicity does not compromise the quality and still allows us to cover 40 languages compared to 9 languages offered by [40].

4.7 Conclusion

We successfully built a multilingual NER system for 40 languages with no language specific knowledge or expertise. We use automatically learned features, and apply language agnostic data processing techniques. The system outperforms previous work in several languages and competitive in the rest on human annotated datasets. We demonstrate its performance on the rest of the languages, by a comparative analysis using machine translation. Our approach yields highly consistent performance across all languages. WIKIPEDIA Cross-lingual links will be used in combination with FREEBASE to extend our approach to all languages.

Chapter 5

Conclusion

5.1 Conclusions

In this thesis we have demonstrated the following:

- The performance speedup obtained in training the current neural network language model is limited by the low compute utilization of the GPU and hence given the current language model, GPU's are not well suited for training them.
- We have investigated some of the structural properties of 2 classes of word embeddings: Polyglot and Skipgram, and demonstrated rich but varied community structure over networks induced on these word embeddings.
- Finally we present a Named Entity Recognition system for over 40 languages using only language agnostic techniques for dealing with noisy training data and no explicit feature engineering. Importantly we were able to demonstrate that using distributed word representations along with language agnostic techniques can be effectively used to learn models which perform competitively on standard datasets when compared models which use a slew of heuristics and language aware techniques.
- We also present a novel technique to evaluate the performance of our models on languages for which no access to a gold standard test data set was available.

5.2 Future Work

One can explore the following research directions from this thesis,which are enumerated below:

- In a recent paper [52] it is shown that computing word embeddings can be significantly simplified by using a technique called Hellinger PCA. It would be interesting to investigate the properties (including network structure) of embeddings learnt through Hellinger PCA and how they compare to Polyglot and Skipgram embeddings.

- Using embeddings trained on corpora from different time periods, one could analyze the network of these word embeddings and seek to investigate how word clusters evolve over time. This could help provide insights into how words change their usage and meaning across time.

- From an engineering standpoint, one could seek to extend the built NER system for more languages and thus scale it up. One can also proceed to analyze the performance of languages which are poor and attempt to derive any language agnostic heuristics that could help boost the performance or identify further systemic errors that contribute to poor performance.

Bibliography

[1] Y. Bengio. Neural net language models. *Scholarpedia*, 3(1):3881, 2013.

[2] The barracuda project. http://seqbarracuda.sourceforge.net/index.html.

[3] D. Randall Wilson and Tony R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Netw.*, 16(10):1429–1451, December 2003. ISSN 0893-6080. doi: 10.1016/S0893-6080(03)00138-2. URL http://dx.doi.org/10.1016/S0893-6080(03)00138-2.

[4] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc Le, Mark Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Ng. Large scale distributed deep networks. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1232–1240. 2012. URL http://books.nips.cc/papers/files/nips25/NIPS2012_0598.pdf.

[5] Bryan Perozzi, Rami Al-Rfou, Vivek Kulkarni, and Steven Skiena. Inducing language networks from continuous space word representations. In Pierluigi Contucci, Ronaldo Menezes, Andrea Omicini, and Julia Poncela-Casasnovas, editors, *Complex Networks V*, volume 549 of *Studies in Computational Intelligence*, pages 261–273. Springer International Publishing, 2014. ISBN 978-3-319-05400-1. doi: 10.1007/978-3-319-05401-8_25. URL http://dx.doi.org/10.1007/978-3-319-05401-8_25.

[6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. 2013.

[7] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost)

from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[9] Holger Schwenk, Anthony Rousseau, and Mohammed Attik. Large, pruned or continuous space language models on a gpu for statistical machine translation. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 11–19. Association for Computational Linguistics, 2012.

[10] Tomas Mikolov, Stefan Kombrink, Lukas Burget, JH Cernocky, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. volume 27, pages 97–110, jun 2011.

[12] Yanqing Chen, Bryan Perozzi, Rami Al-Rfou', and Steven Skiena. The expressive power of word embeddings. In *ICML 2013 Workshop on Deep Learning for Audio, Speech, and Language Processing*, volume abs/1301.3226, Atlanta, USA, 2013.

[13] Rami Al-Rfou', Bryan Perozzi, and Steven Skiena. Polyglot: Distributed word representations for multilingual nlp. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 183–192, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W13-3520`.

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[15] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? In *Database TheoryICDT99*, pages 217–235. Springer, 1999.

[16] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT*, pages 746–751, 2013.

[17] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[18] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks, 2009. URL http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154.

[19] David Eppstein, Michael S Paterson, and F Frances Yao. On nearest-neighbor graphs. *Discrete & Computational Geometry*, 17(3):263–282, 1997.

[20] Ramon Ferrer i Cancho and Richard V Solé. The small world of human language. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 268(1482):2261–2265, 2001.

[21] Chris Biemann, Stefanie Roos, and Karsten Weihe. Quantifying semantics using complex network analysis. In *Proceedings of COLING 2012*, pages 263–278, Mumbai, India, December 2012. The COLING 2012 Organizing Committee. URL http://www.aclweb.org/anthology/C12-1017.

[22] Lucas Antiqueira, Osvaldo N Oliveira Jr., Luciano da Fontoura Costa, and Maria das Graças Volpe Nunes. A complex network approach to text summarization. *Information Sciences*, 179(5):584–599, 2009. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2008.10.032. URL http://www.sciencedirect.com/science/article/pii/S0020025508004520.

[23] Jean Véronis. HyperLex: lexical cartography for information retrieval. *Computer Speech & Language*, 18(3):223–252, 2004. ISSN 0885-2308. doi: http://dx.doi.org/10.1016/j.csl.2004.05.002. URL http://www.sciencedirect.com/science/article/pii/S0885230804000142.

[24] Adilson E. Motter, Alessandro P. S. de Moura, Ying-Cheng Lai, and Partha Dasgupta. Topology of the conceptual network of language. *Phys. Rev. E*, 65:065102, Jun 2002. doi: 10.1103/PhysRevE.65.065102. URL http://link.aps.org/doi/10.1103/PhysRevE.65.065102.

[25] Mariano Sigman and Guillermo A Cecchi. Global organization of the wordnet lexicon. *Proceedings of the National Academy of Sciences*, 99(3): 1742–1747, 2002.

[26] Geoffrey E Hinton. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, pages 1–12. Amherst, MA, 1986.

[27] DE Rumelhart, GE Hinton, and RJ Williams. Learning internal representation by back propagation. *Parallel distributed processing: exploration in the microstructure of cognition*, 1, 1986.

[28] Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning*, pages 137–186. Springer, 2006.

[29] Yoshua Bengio and J-S Senecal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *Neural Networks, IEEE Transactions on*, 19(4):713–722, 2008.

[30] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252, 2005.

[31] Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088, 2008.

[32] Xavier Carreras, Lluís Màrques, and Lluís Padró. Named entity extraction using adaboost. In *Proceedings of CoNLL-2002*, pages 167–170. Taipei, Taiwan, 2002.

[33] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 168–171. Edmonton, Canada, 2003.

[34] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2078186.

[35] J. Turian, L. Ratinov, and Y. Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[36] Joel Nothman, Tara Murphy, and James R. Curran. Analysing Wikipedia and gold-standard corpora for NER training. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages

612–620, Athens, Greece, March 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/E09-1070.

[37] A. Toral and R. Munoz. A proposal to automatically build and maintain gazetteers for Named Entity Recognition by using Wikipedia. *Proceedings of the EACL-2006 Workshop on NEW TEXT-Wikis and blogs and other dynamic text sources*, 2006. URL http://acl.ldc.upenn.edu/W/W06/W06-2809.pdf.

[38] Jun'ichi Kazama and Kentaro Torisawa. Exploiting Wikipedia as external knowledge for named entity recognition. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 698–707, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D/D07/D07-1073.

[39] Alexander E Richman and Patrick Schone. Mining wiki resources for multilingual named entity recognition. In *ACL*, pages 1–9, 2008.

[40] Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, and James R Curran. Learning multilingual named entity recognition from wikipedia. *Artificial Intelligence*, 194:151–175, 2013.

[41] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*, 2008.

[42] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 1:213, 2002.

[43] Léon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2. URL http://leon.bottou.org/papers/bottou-91c.

[44] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 999999:2121–2159, 2011.

[45] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages

1247–1250, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6. doi: 10.1145/1376616.1376746. URL http://doi.acm.org/10.1145/1376616.1376746.

[46] Carla E. Brodley and Mark A. Friedl. Identifying mislabeled training data. *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH*, 11: 131–167, 1999.

[47] Xingquan Zhu, Xindong Wu, and Qijun Chen. Eliminating class noise in large datasets. In *ICML*, volume 3, pages 920–927, 2003.

[48] Erik F. Tjong Kim Sang. Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158. Taipei, Taiwan, 2002.

[49] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2003*, pages 142–147, 2003.

[50] Joel Nothman, James R Curran, and Tara Murphy. Transforming wikipedia into named entity training data. In *Proceedings of the Australasian Language Technology Association Workshop 2008*, pages 124–132, Hobart, Australia, December 2008. URL http://www.aclweb.org/anthology/U08-1016.

[51] Sungchul Kim, Kristina Toutanova, and Hwanjo Yu. Multilingual named entity recognition using parallel data and metadata from wikipedia. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 694–702, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P12-1073.

[52] Rémi Lebret and Ronan Lebret. Word emdeddings through hellinger pca. *CoRR*, abs/1312.5542, 2013.